Gruppe 1: 18/10-21

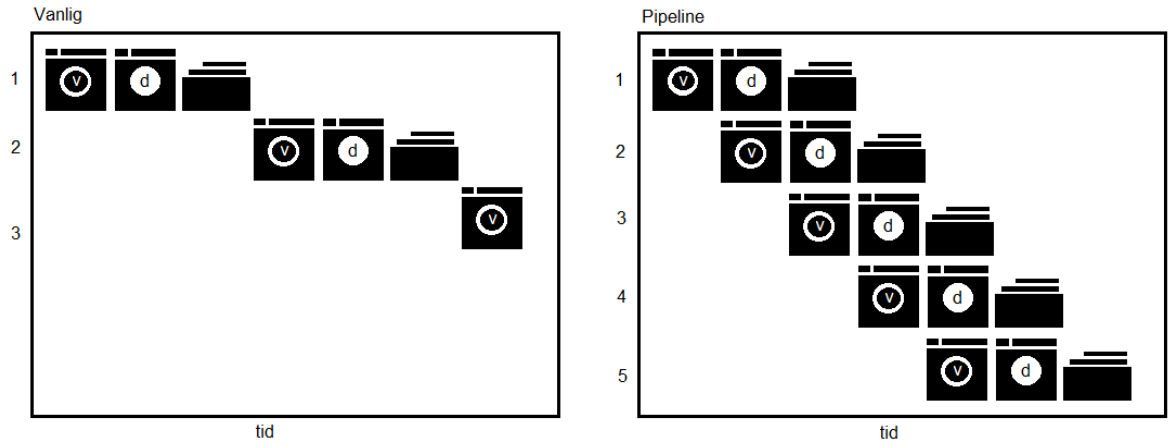
Vi husker at gårsdagens maskiner brukte reléer og vakuumrør i stede for transistorer og skulle kanskje tro at det å lage transistorer som switcher raskere og raskere er nok for å lage en rask maskin. Vi har også snakket om klokkehastighet og kretser, som begge kan brukes til å lage raskere datamaskiner. I dag ser vi på bus-er, minne og pipeline.

Bus:

Før i tiden pleide man å ha RAM i separate kabinetter. Dataen kunne fraktes mellom kabinettene og CPU-en via “bus-bars", som senere ble kjent som en bus. I dag er RAM og CPU i samme device, men bus-er er fremdeles relevante for å frakte data mellom dem. Dette gjøres ofte ved å dele opp data i mindre biter, som fraktes gjennom flere “lanes” (ledninger) inni en bus. Et sett med data kan altså fraktes gjennom en bus over flere klokkesykluser, hvis det trengs. Noen bus-er bruker bare en ledning og frakter en bit med data av gangen over denne. Dette skjer gjerne over lenger avstander.

Pipeline:

Pipeline er fint illustrert gjennom vaskemaskin, tørketrommel, brette-eksemplet. Se for deg at du eier et hotell, men har bare en vaskemaskin, en tørketrommel og du er alene når du bretter vasken. Her skal mange laken vaskes, tørkes og brettes. Du setter på en vask, venter, tar ut tøyet, setter på en tørk, venter, tat ut tøyet og bretter vasken. Så starter du på nytt. Dette tar tid... En mer effektiv måte å gjøre det på er å starte en ny vask med en gang du har tatt ut den første av vaskemaskinen. Når tørketrommelen er ferdig er vask nr. 2 også ferdig og kan flyttes fra en maskin til den neste. Nå er vaskemaskinen ledig for vask nr. 3... osv. Tre ganger så raskt!



I datamaskinen vår kan man oversette denne situasjonen til at CPU-en skal utføre fetch (hent), decode (hva den skal gjøre), execute (utfør) og write back (hva skjedde), mange ganger. Da er det raskere å gjøre dette som en pipeline enn å gjøre fetch, decode, execute, write back, fetch,

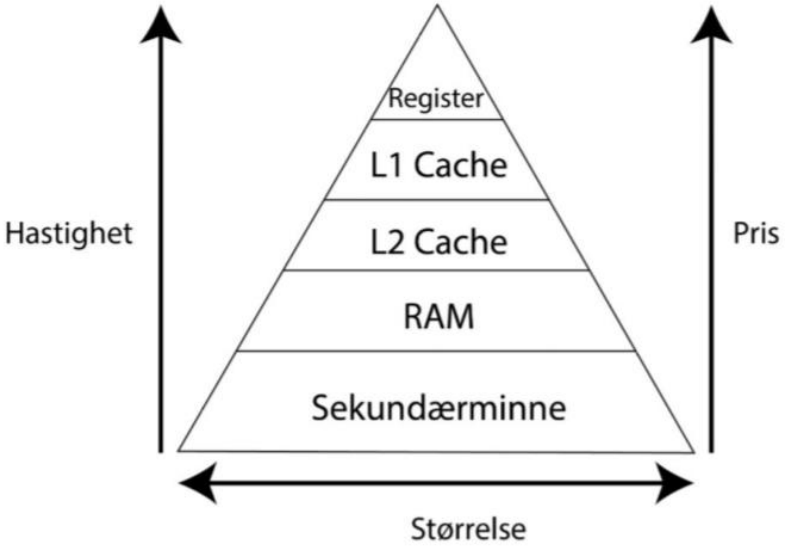
decode, execute, write back, fetch.. Osv.. Osv. Med hjelp av en pipeline går det altså fire ganger så raskt, og maskinen klarer å fetche, decode, execute og write back på en gang i én klokkesyklus!

Hazards

Ulempen med dette er at man potensielt kan støte på en *hazard*. En data hazard skjer hvis data som bearbeides er avhengig av hverandre på en slik måte at den ikke blir bearbeidet ferdig i tide når den skal brukes. Dette trenger vi ikke å tenke på i vaske-eksemplet, men i datamaskinen er ofte data en del av en større sammenheng, så det er dumt hvis vi skal begynne å fetche noe som ikke er klart. Et laken var ikke rent allikevel, men du får ikke satt laken 1 inni vask 2, for den har allerede begynt. En strukturell hazard er når mer enn en instruks prøver å bruke en ressurs. I vårt eksempel blir det som å prøve å vaske og tørke laken på en gang.

Minne:

Her er illustrasjonen på minne-hierarkiet fra forelesningen



HDD/SSD Hard Disk Drive og Solid State Drive er eksempler på sekundærminne og det er her hele programmet er lagret. Data som lagres her forsvinner ikke når man skrur av maskinen, i motsetning til alle de andre over. Sekundærminne har også størst kapasitet av alle minne-kategoriene. Hard disker har bevegelige deler inni seg. De er altså mekaniske, som gjør at de er i ferd med å bli utdaterte. De fleste nye maskiner bygges med solid state drive, som bruker flash-minne. Flash er enda en type sekundærminne.

RAM husker vi fra LMC. Det er random access memory og kan holde på både instruksjoner og data. RAM er ganske raskt. De første SSD-ene ble laget med RAM. Ulempen med dette var at data forsvinner fra RAM når man skrur av maskinen.

Cache er svært nyttig når man vil lage en rask maskin. Vanligvis når vi henter data vil CPU-en hente en og en bit fra RAM. Dette kan ta titalls klokkesykluser og kaster bort mye tid CPU-en kunne brukt på å bearbeide informasjon. Løsningen på dette problemet er å sette en liten bolk med minne nær CPU-en. Dette er cache. Når CPU ber om data fra RAM henter den ikke bare en bit, men en bolk med bits som lagres i cache. Neste gang den ber om data sjekker den cache først. Hvis daten er i cache er dette en cashe-hit, som betyr at CPU-en får tak i denne dataen bare på en klokkesyklus i stede for titalls, som vanlig. Hvis ikke dataen er i cache er dette en cache-miss og vi må gå veien om RAM allikevel. Cache kan også brukes som midlertidig lagring for utregninger, slik at CPU slipper å gå frem og tilbake mellom RAM for å hente data den egentlig ikke var ferdig med å bruke. Datamaskinen synkroniserer gjerne cache og RAM når cache er full men mer data skal hentes fra RAM. Dette gjør den for at ikke endringer gjort på data via cache går tapt, men lagres i RAM. Jeg fikk beskjed om å halle cache-hit og –miss RAM-hit og RAM-miss. Det er det samme, bare med andre ord.

Register husker vi fra LMC. De er minnekomponenter inni CPU-en som holder på midlertidig data. De er veldig raske, men vi har som regel relativt få av dem siden de er dyre.

Flop-flop er ikke med i modellen eksplisitt, men den er med som følge av at bl.a. RAM er med. Det er en grunnleggende komponent for å lage minne. De holder 1 bit med data.